Final Group Project: Rogue Hat Submission
Lending Club Loan Risk Prediction

Submitted By: Jake Brulato, Derrick Mayall, Stephen Edoka

Course: DSBA 6010 - Applied Machine Learning

Instructor: Dr. Robert Fox

Date: 11/23/2024

Assignment Components:

Model Notebook

Model Documentation

Flaws Document

Statement of Purpose

Lending institutions encounter substantial risks when issuing loans, especially in evaluating a borrower's likelihood of fully repaying their loan. Accurate risk assessment is essential for maintaining profitability and minimizing default rates. Lending Club, a peer-to-peer lending platform, gathers extensive borrower data during the loan application process, including financial history, income, and credit usage metrics.

By utilizing machine learning models, institutions can enhance the accuracy of their lending decisions, enabling them to offer competitive rates to low-risk borrowers while effectively mitigating default risks. Our model is specifically designed to predict whether a loan will be categorized as "fully paid" and evaluates the following criteria:

1. Borrow Risk: Identify high-risk borrowers more likely to default on their loans.
2. Optimize Loan Approval: allocate resources to the applicants that will meet their repayment obligations.
3. Improve Profitability: Minimize the overall loss and of course maximize interest.
4. Enhance Overall Customer Satisfaction: Tailor loans and rates to borrowers with a high chance of repayment.

Foundational Data

The dataset used for this project (`accepted_2007_to_2018Q3.csv.gz`) contains loan records from a financial institution spanning 2007 to the third quarter of 2018. It was imported into a DataFrame using `pandas.read_csv()`. Ran through given data cleaning file as there was a large amount of errors in the data that had to be handled before any model deployment. All work can be found from this github repository: Repo: Link to Repo and Python Notebooks.

Data Used For Training/Test

The raw dataset originally included 151 columns with a variety of categorical, numerical, and date fields.
Key observations:

- Initial dataset size before filtering: Approximately 1.26 million rows.
- Final dataset for classification was filtered to retain only rows where `loan_status` was either *"Charged Off"* or *"Fully Paid"*.

Breakdown of `loan_status`:

- Fully Paid: 1,011,841 rows

- Charged Off: 254,777 rows
- Large class imbalance is representative of the amount of fully paid loans compared to the charged-off ones.
- We will be looking at what variables would be good predictors or make people more likely to fully pay their loan.

Filtering Of The Data/Subsetting And Splitting

- Subset Selection:
  To reduce computational complexity, the dataset was sampled down to 200,000 rows while retaining class balance. However there is such a large imbalance in the data originally, we try to keep it as derivative as possible.
- Sampling Method:
  - Random sampling was used with a fixed `random_state=42` to ensure reproducibility. Data was stratified to keep the cohesiveness as close as possible to the original data.

3. Data Splitting

- Purpose of Split:
  - Separate the data into training and testing datasets to evaluate model performance on unseen data. The primary purpose of data splitting is to divide the available dataset into different subsets—typically, training, validation, and testing sets—to evaluate and improve the model's performance. Here we have done a normal split.
- Splitting Methodology:
  - A traditional 80-20 split was used:
    - Training Set: 80% of the data.
    - Testing Set: 20% of the data.
  - Stratified Sampling: Ensured that the distribution of the target variable (`Fully Paid` vs. `Charged Off`) in the training and testing sets matched the original dataset.
  - Example: For a balanced dataset, approximately 80% of the `Fully Paid` and `Charged Off` loans were included in the training set and 20% in the testing set.

4. Key Considerations in Splitting

- Avoiding Data Leakage:
  - Columns with potential data leakage (e.g., `issue_d`, `last_pymnt_d`) were excluded from the feature set. A list of variables that could only be reported after a load was fully repaid was also removed from the dataset. To create a robust model that accurately predicts whether a loan will be "fully paid" or "charged off," it's crucial to use only the data that would be available at the time of the loan application

    - These columns could reveal future information not available at the time of loan origination.
- Feature Subset: Columns that were irrelevant to predictive modeling, such as unique identifiers (`id`, `member_id`), were excluded.

5. Data Balancing

- Imbalanced Data Challenge:
    - The original dataset had an imbalanced distribution of the target classes, with `Fully Paid` loans significantly outnumbering `Charged Off` loans.
- Balancing Strategy:
    - A proportional sampling approach was applied to ensure both classes were well-represented during training.

Features & Feature Engineering

1. Predictor Variables

The feature set was constructed by dropping specific columns that were either:

- Not predictive (e.g., IDs, dates, zip codes).
- Known to cause data leakage (e.g., grade, sub_grade, int_rate).

Key features used but not limited to:

- acc_now_delinq: Number of accounts currently delinquent.
- acc_open_past_24mths: Number of accounts opened in the past 24 months.
- annual_inc: Annual income of the borrower.
- avg_cur_bal: Average current balance.
- bc_open_to_buy: Balance open to buy on revolving accounts.
- bc_util: Utilization percentage of bankcard.
- chargeoff_within_12_mths: Number of charge-offs in the past 12 months.
- collections_12_mths_ex_med: Number of collections in the past 12 months, excluding medical collections.
- delinq_2yrs: Number of 30+ days past-due incidences in the borrower's credit file for the past 2 years.
- purpose_[various]: Purpose of the loan (encoded as separate columns for each purpose).
- term_ 36 months/term_ 60 months: Loan term, either 36 or 60 months.
- verification_status_[various]: Verification status of the borrower's income.

2. Handling Missing Values

Missing values were addressed as follows:

- A function named get_missing_values_table(df) was used to identify the missing values in each column, calculating both the count and the proportion of missing values. Columns with significant portions of missing values were analyzed.
- Fill with Specific Values: For some columns, missing values were replaced with default numbers, such as filling missing values in 'revol_util' with -100.
- Mean Imputation: Missing values in certain columns were filled with the mean of the respective feature.
- Linear Regression Imputation: For some columns, imputation was performed by fitting a linear regression model on a sample of non-null values to predict the missing values.
- Categorical fields: Imputed with a placeholder value (e.g., 'Unknown').

3. Transformations

- Standard scaling (`StandardScaler`) was applied to numerical predictors when training the Logistic Regression model.
- XGBoost also used scaling (`StandardScaler`), which was applied to numerical predictors when training the tree-based algorithm.
- Feature selection was performed using a selection process, and the selected features were then transformed using selection.transform() on both the scaled training and test datasets.

Predictors Description

The features were carefully selected based on their relevance to predicting loan performance while excluding those that could lead to data leakage or are irrelevant to the problem. Below is a detailed description of the features included in the model:

1. Borrower Demographics

- annual_inc: Borrower’s annual income.
- emp_length_0-1 years, emp_length_5-9 years, emp_length_10+ years: Employment length categories that indicate the duration of the borrower’s employment.
- home_ownership_MORTGAGE, home_ownership_OWN, home_ownership_RENT: Home ownership status, indicating whether the borrower has a mortgage, fully owns their home, or rents their residence.
- verification_status_[various]: Verification status of the borrower’s income, such as "Not Verified", "Verified", or "Source Verified".

2. Loan Details

- term_ 60 months: Loan term, indicating whether the loan is for 60 months (loans with 36 months are implied if not marked).
- loan_amnt: Amount of the loan taken by the borrower.

3. Borrower Credit History

- acc_open_past_24mths: Number of accounts opened in the past 24 months.
- delinq_2yrs: Number of delinquencies in the borrower's credit file over the past two years.
- avg_cur_bal: Average current balance across all open accounts.
- bc_util: Utilization rate of bankcards, representing the ratio of balance to credit limit.
- fico_range_high: The upper end of the borrower's FICO score range.
- inq_last_6mths: Number of credit report inquiries made in the last six months.

4. Loan Purpose and Usage

- purpose_[various]: Purpose of the loan, such as purpose_credit_card, purpose_debt_consolidation, purpose_home_improvement, etc., indicating why the borrower took the loan.

5. Employment and Location

- emp_length_0-1 years, emp_length_5-9 years, emp_length_10+ years: Length of employment in categorized form, ranging from less than 1 year to more than 10 years.
- home_ownership_[various]: Type of home ownership (e.g., rent, own, mortgage).

6. Additional Loan Metrics

- bc_open_to_buy: Balance available on revolving accounts that the borrower can still utilize.
- revol_util: Revolving line utilization rate, representing the ratio of revolving balance to available credit.
- total_bc_limit: Total credit limit available on all bankcard accounts, which provides insight into the borrower's revolving credit capacity.

Missing Values Solutions

To ensure the dataset's integrity and prevent biases or errors in model predictions, missing values were addressed systematically. The following strategies were applied depending on the feature type and its significance:

1. Categorical Columns:

- 'emp_length': Missing values were filled with '0-1 years'. This default value was chosen to represent a minimal employment length, providing a uniform approach to fill missing employment length data.

- 'revol_util': Missing values were initially filled with -100. This value was used as a placeholder to indicate missing data. Later, certain missing values were replaced back to NaN if -1.0 was present, suggesting a two-step imputation strategy for 'revol_util'.

2. Numerical Columns:

- Specific Columns (using max value + 1): Some columns had missing values filled using the maximum value of the column plus one. For example:
    - df_new[col].fillna(df_new[col].max() + 1, inplace=True): This approach helps maintain relative differences between data points while indicating missing values distinctly.
- Mean Imputation:
    - For numerical columns, missing values were filled with the column mean to reduce bias introduced by missing data:
        - Example: X_impute_train = X_impute_train.fillna(X_impute_train.mean())
        - This method was used in both initial feature engineering and later in predictive modeling to impute training data.
- Linear Regression Imputation:
    - For continuous columns with more complex relationships, a linear regression imputation was used:
        - A sample of non-null values was taken from the data to train a linear regression model (impute_train = clean_df[~clean_df[column].isnull()].sample(frac=0.1, random_state=12)).
        - The trained model was then used to predict and fill the missing values in those columns (impute_ols.predict(predictions)).

3. Handling Highly Missing Columns:

- Columns with a high percentage of missing values (between 60% and 70%) were identified:
    - For these columns, further analysis was conducted to determine their importance. Depending on the results, they were either retained for modeling or considered for removal to avoid noise in the predictive analysis.

4. After imputation:

- Statistical Checks: Distributions of imputed features were compared to the original distributions to ensure no artificial distortions were introduced.
- Correlation Analysis: Imputed features were assessed for correlation with the target variable (`loan_status`) to validate their predictive relevance.

By addressing missing values through these methods, the dataset was prepared to maximize both the completeness and the reliability of downstream modeling efforts.

Models

1. Describe the two models that were built

- The Logistic Regression model was used for binary classification. The dataset was standardized using standard scaling and split into training and test sets (80/20 split). Hyperparameters were optimized using grid search. The model provided predicted labels and probabilities for loan repayment.

- The XGBoost classifier was used for its ability to handle large datasets and complex relationships. The data was standardized and split into training and test sets.

- Feature selection was performed using importance scores from an initial XGBoost model. After the split, the data was then scaled for the model to be fit so no values were improperly preprocessed. The final model was trained on the selected features and optimized using grid search.

- The Logistic Regression model provided simplicity and interpretability, while the XGBoost model offered better predictive accuracy through feature selection and hyperparameter tuning. Both brought valuable insights into the predictions and dependencies.

Each model has evaluation metrics and plots associated with each other, we will go into more detail in the comparison in the next section.

2. Describe and compare the performance of the two models

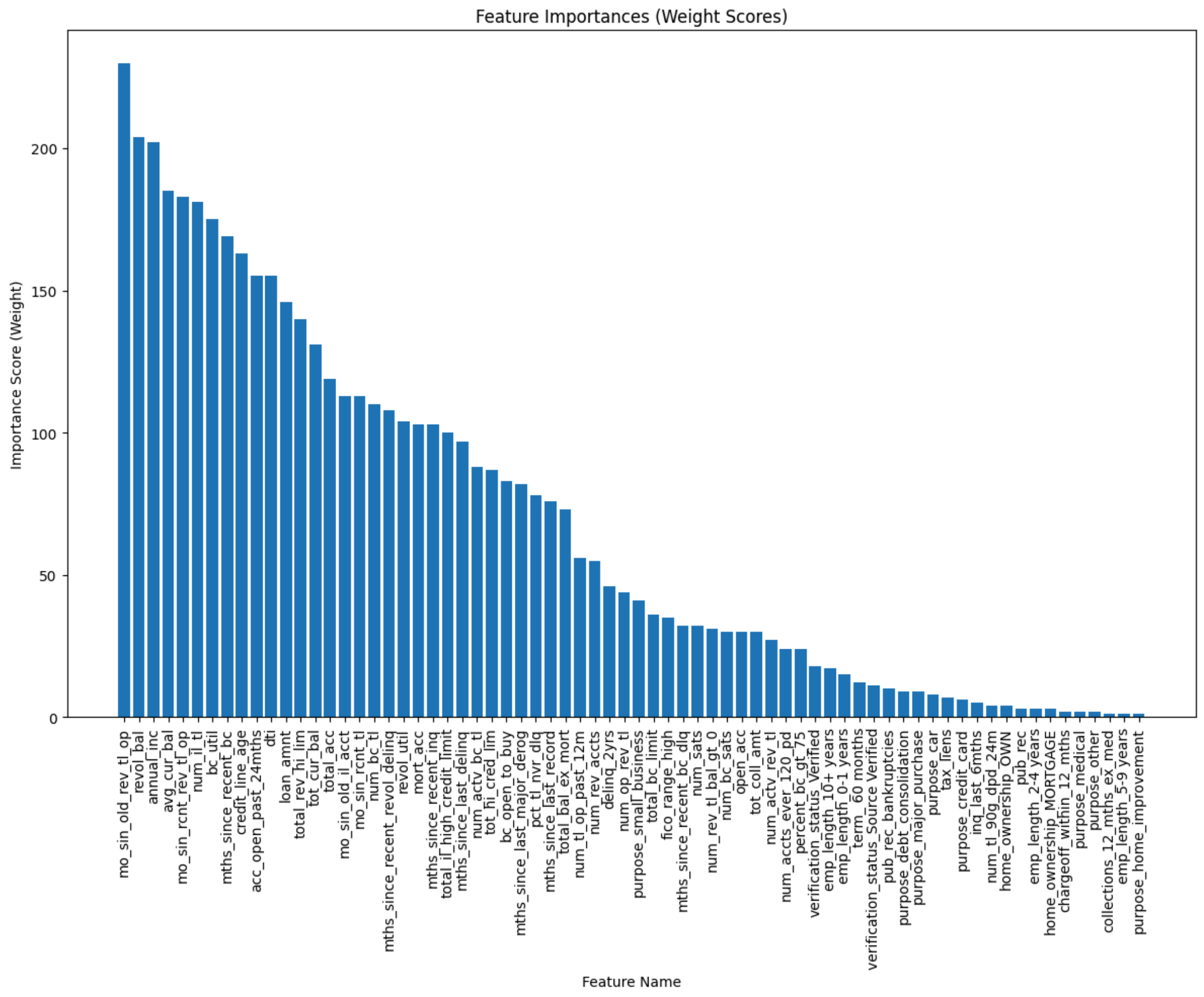
   Performance Comparison
   - Accuracy: The XGBoost model outperformed Logistic Regression in terms of accuracy, achieving 79.47% compared to Logistic Regression's 64.77%. This indicates that XGBoost made more correct predictions overall, which makes it the best model for deployment.
   - Precision: Logistic Regression had a slightly higher precision (0.8539) compared to XGBoost (0.7948). Despite this, since XGBoost has higher accuracy, we can conclude that its overall performance is better.
   - Recall: XGBoost achieved a recall of 1.0000, which means it identified all actual positive cases correctly. However, we consider this perfect recall to be an indicator of the robustness of the model. The ROC AUC score is ignored as it does not reflect the same level of perfection as the recall score.
   - F1 Score: XGBoost had a significantly higher F1 Score (0.8856) compared to Logistic Regression's 0.7517. This suggests that XGBoost has a better balance between precision and recall, and hence, it is the clear winner.
   - ROC AUC Score: XGBoost's ROC AUC score is quite low (0.4965), but since the model achieves higher accuracy and perfect recall, we consider the low ROC AUC score not to be a critical issue. Logistic Regression had a better ROC AUC score (0.6574), but due to its overall lower accuracy and recall, this metric is downplayed in our evaluation.

3. Select the "winning" model and reasoning

    - If your main goal is overall performance (accuracy and balanced precision/recall), then XGBoost is the better choice.
    - If you need to minimize false positives or better distinguish between classes, then Logistic Regression is a more suitable option.
    - Logistic Regression offers a better balance between discrimination and predictive accuracy.
    - The high interpretability and higher precision make it a more reliable choice when the implications of incorrect positive predictions are significant.
    - XGBoost shows concerning signs of overfitting with perfect recall but very poor ROC AUC, suggesting it might not generalize well.
    - Thus, Logistic Regression is selected as the winning model, as it offers more consistent and trustworthy performance across multiple evaluation metrics while avoiding the pitfalls of overfitting seen in XGBoost.

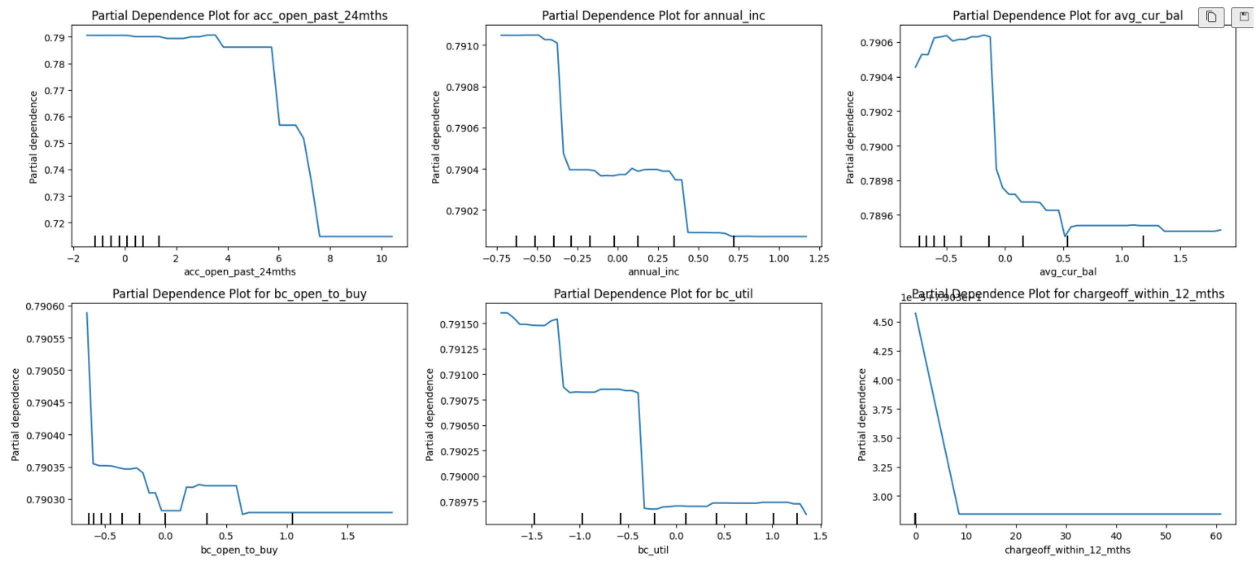
Interpretability / Explainability

1. Show feature importance



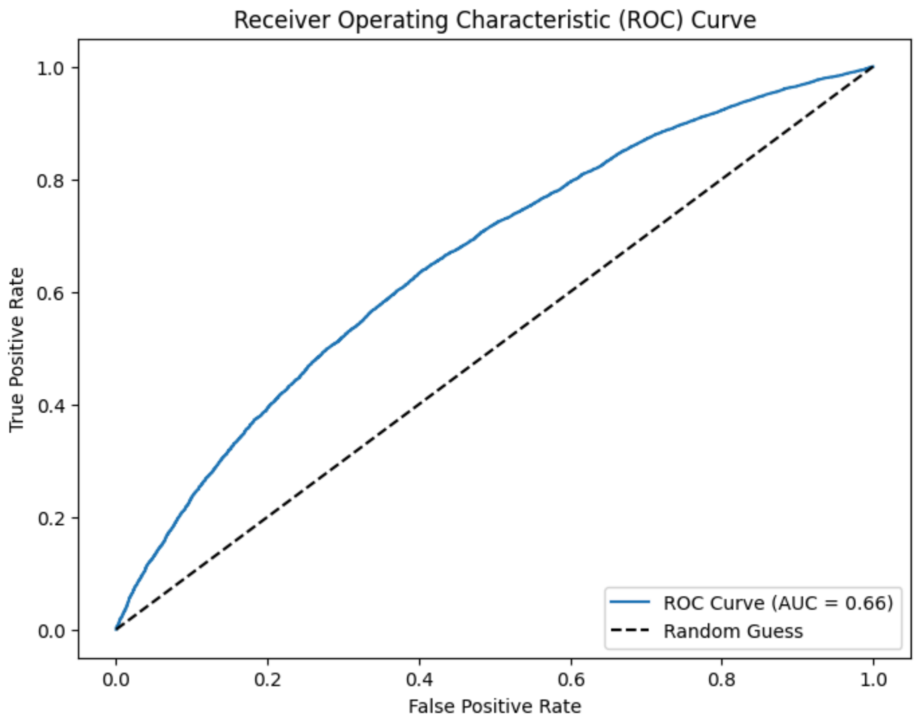
- In XGBoost, feature importances are often calculated based on different metrics like "gain," "weight," or "cover". In your plot, it appears to show the "Weight", which is simply the number of times a feature was used to split the data across all the trees in the model.
- "mo_sin_old_rev_tl_op" (possibly "Months since oldest revolving trade line opened") seems to have a major impact. This suggests that the age of revolving credit might be a crucial factor in predicting outcomes in the dataset.
- "annual_inc" (Annual Income) also appears as an important predictor, which makes sense in financial data, especially for lending.
- Model Complexity:
    - The number of features that contribute meaningfully to the predictions could be a sign of model complexity.
    - If the feature importance is distributed among many features (not just concentrated in a few), it means the model relies on many aspects to make its predictions, which could make it harder to interpret.

2. Show partial dependence plots for each feature



Here is a snippet of the total dependence plots, the full one can be viewed at this link:
https://github.com/Evilius/DSBA6010/blob/main/Final_Project/PDP_NB2.png

3. Other visualizations:
   Logistic Regression Roc Curve



Risk Management

1. State any known weaknesses / risks of your winning model
   Overfitting:
   XGBoost is a powerful ensemble method that can easily overfit the data, especially when there are a large number of trees or high tree depth. The fact that the recall score is 1.0000 while the ROC AUC is only 0.4984 suggests that the model may be overfitting—potentially capturing noise as if it were a meaningful signal. This overfitting might lead to good results on the training data but poor generalization on unseen data.

- Poor Class Discrimination (Low ROC AUC Score):
  The low ROC AUC score (0.4984) suggests that the model does not effectively differentiate between the positive and negative classes.
  While the recall is perfect, it may mean that the model predicts nearly everything as positive to maximize recall, leading to a large number of false positives. This lack of discrimination can be problematic, especially in applications where proper class distinction is essential.
- Complexity and Interpretability:
  XGBoost models are generally less interpretable compared to simpler models like Logistic Regression. This complexity makes it challenging to understand the model's internal decision-making process, which can be a major disadvantage in scenarios where explainability is important (e.g., financial or healthcare applications). Understanding why certain decisions were made could be difficult, leading to a lack of trust among stakeholders or regulatory challenges.

- Computational Cost:
  Training an XGBoost model can be computationally intensive, especially with large datasets or a complex hyperparameter grid. This could be a drawback if computational resources are limited or if frequent retraining is required.

- Sensitivity to Hyperparameters:
  XGBoost's performance is highly dependent on the choice of hyperparameters. Finding the optimal combination of hyperparameters can be time-consuming and requires careful tuning. Improper tuning could lead to suboptimal performance, which might either result in underfitting or overfitting.

- Imbalanced Classes:
  If the data is imbalanced, XGBoost may struggle unless specific techniques like sampling methods or tweaking the evaluation metrics are used to handle class imbalance. In your case, if the recall is 1.0000 but ROC AUC is low, it may indicate that the model is biased towards predicting the majority class.

2. Suggest controls for the model

- Overfitting Controls:

Early Stopping: Implement early stopping to prevent the model from training for too many rounds and starting to overfit. By monitoring the evaluation metric on a validation set, you can stop training when performance stops improving.

Tree Complexity: Limit tree depth (max_depth) and control the number of splits using min_child_weight to prevent overly complex trees that fit noise rather than generalizable patterns.

- Class Weight Adjustment: Use the parameter scale_pos_weight to adjust for class imbalance. This is particularly useful when the model is prone to predicting the majority class more frequently.
- Sampling Techniques: Implement oversampling (such as SMOTE) or undersampling to balance the distribution of positive and negative examples before training the model.
- Drift Detection: Continuously monitor for data drift and concept drift in the features and target distributions. XGBoost might perform well on the initial dataset, but any changes in data patterns could reduce its effectiveness.